HAYDEN Macintosh Library BOOKS

48431 Available on Disk

The Waite Group's Tricks of the HyperTalk™ Masters

Foreword by Steve Maller

Edited by
The Waite Group

The Waite Group's

Tricks of the HyperTalk™ Masters

Chapter 10 Synching Sight and Sound
From Sound Effects to Music Videos 249

Part 5 Tools and Utilities 267

Chapter 11 Developer's Tool Kit
Utilities and Scripts for Better Scripting 269
Chapter 12 The Quiz Maker
Create Self-Administering and Self-Grading Exams 311
Chapter 13 HyperAnimals
A Game That Learns from the Player 355
Chapter 14 A Grab Bag of Utilities
Snapshots, Printing, Dialer, and More 389

Part 6 Communications 413

Chapter 15 Chauffeur
A HyperCard-to-CompuServe Front End 415

Part 7 Performance 449

Chapter 16 Benchmarking HyperTalk
Unveiling HyperTalk Performance 451
Chapter 17 Large Stacks
Speeding Up Large Stacks 491
Chapter 18 Extending HyperTalk with Externals
Filling "Holes" with XCMDs and XFCNs 519

Part 8 Reference 567

The Complete HyperTalk Reference
Everything to Script With and More 569

Contents

Foreword xxv

Preface xxvii

Acknowledgments xxix

The Authors xxxi

Part 1 General Tips and Tricks 1

1 ***Techniques for Stack Development*** 3
Notes from the Scripting Battlefields
—Jeanne A. E. DeVoto

Perspectives on HyperTalk 5
Mastering Messages 6
Handlers 6
The Message Path 8
Local and Global Variables 9

The HyperTalk Zoo 10
The *it* Variable 11
me and *the target* 11
the selection 12
the result 13
the 13

Advanced HyperTalk Topics 14
Limiting the *find* Command 14

Nick Hodge is the Technical Support Manager for an Apple Dealership in Adelaide, South Australia. A Macintosh programmer since Macintoshes appeared in South Australia, and a frustrated Pascal programmer, he now finds HyperTalk a perfect way to write programs. Mr Hodge is the youngest Macintosh master in this book and has produced the largest project stack. His Chauffeur front-end consumes 106 kilobytes and its listings require 65 pages. He is the author of chapter 15, "Chauffeur."

Ted Jones is a professional programming consultant and author specializing in Microsoft BASIC and HyperTalk. He is co-author of *The Waite Group's HyperTalk Bible* and a contributor to the BMUG (Berkeley Macintosh Users Group) Newsletter. Mr. Jones has a bachelor of science in Liberal Arts from Arizona State University. Along with Mitchell Waite, Mr. Jones is the creator of PolyButtons, which expands HyperTalk by adding multiple-sided, nonrectangular buttons to the scriptor's toolbench. Mr. Jones assembled the companion stack to this book and to *The Waite Group's HyperTalk Bible*. He is the author of chapter 4, "Quick Pop-Up Fields," Chapter 14, "A Grab Bag of Utilities," and the Reference section, and the co-author of chapter 3, "Dueling Scroll," and chapter 6, "PolyButtons."

Steven F. Martin works at the Computer Science Center of Texas Instruments in Dallas, Texas, where his current work in the field of Hypermedia exposed him to using HyperCard to manage very large databases. Mr. Martin has a masters of science degree in Electrical Engineering and Computer Science from The Massachusetts Institute of Technology. His love of cinema led to an ambitious (more than 5 Mbytes) database to catalog movies and gave him many insights into speeding up large stacks. He is the author of chapter 17, "Large Stacks."

Jack A. Smith is a staff engineer for the Applied Mathematics and Process Simulation group of Central Research and Engineering at Union Carbide. He specializes in system software development for UNIX-based engineering workstations and in computational chemistry, particularly molecular modeling. He holds a master of science degree and a doctorate in Chemical Physics from the University of Florida and was a member of that University's Quantum Theory Project. His most recent work includes scientific application development on the Macintosh using C and Fortran. His Menu Fields scripts show adaptive techniques for extending HyperCard's capability of making custom menus. He is the author of chapter 5, "Menu Fields."

Andrew Stone is a Macintosh programming consultant who enlisted in the computer revolution in 1980. An architect and builder, he now develops custom software in C, Fortran, and HyperTalk for clients. Mr. Stone is the "Prime Minister" of New Mexico's largest HyperCard Special Interest Group and is a prolific scriptor. In his two chapters, he provides a battery of scripting tools that he has developed over the last year, including the QuizMaker, an invoice maker, and a stack reporter. He is the author of chapter 11, "Developer's Tool Kit," and chapter 12, "The Quiz Maker."

Chapter 15

Chauffeur

A HyperCard-to-CompuServe Front End

Nick Hodge

Synopsis: Bill Atkinson's original vision of HyperCard was a product that would turn a "cold" hard-to-use text based medium into a "warm" graphics-oriented interface that nontechnical people could easily use. To this end, he provided all tools to create such an interface. Unfortunately, designing and setting up such a front end is not a trivial task. For example, a vast amount of data exists in text format on dozens of popular on-line databases, such as Compuserve. HyperCard provides no easy way to manipulate the data on these services.

This chapter presents Chauffeur, a front end that completely insulates you from Compuserve's dry text-based interface. Chauffeur automatically dials Compuserve from anywhere in the world, gets all your waiting EasyPlex electronic mail, talks to people in Forums, delivers any mail you wish to send, scans for messages from users, scans for files to download, and even keeps an address book of all your contacts. The detailed chapter reveals tricks for serial port control, has enhanced serial XCMDs, shows hiding passwords, and much more. You can add sampled sounds so that Chauffeur talks to you as it moves to different areas of Compuserve, and you can learn all the requirements of any general purpose front-end from this chapter. There is a built-in help feature and even an AutoPilot mode that learns and records your manual activities so that Chauffeur can follow them later.

HyperCard has simplified the creation of easy-to-use information retrieval engines to the point where a stack author can almost transform information into knowledge.

Current electronic information databases are available on CDROMs and in vast electronic networks such as CompuServe and The Source. To use these systems, you need to master a complex set of commands. Or you can use HyperCard to create a *front end*—a user interface that is placed over the front of a less friendly system.

This chapter describes the process involved in creating a front end that interfaces with CompuServe using HyperCard. Bill Atkinson has called HyperCard a "user interface toolkit," and this description can be applied as well to any stack created with HyperCard. Chauffeur is one of the first of many front ends written in HyperCard.

What Is a Front End?

As a user interface, a front end runs in front of another program or system that is usually more difficult to use. One example of a front end is the Finder. It simplifies the tasks of copying files, erasing disks, and renaming directories. Without the Finder, each Macintosh user would have to remember a complex set of commands, such as those used in MS-DOS and UNIX. For example, to create a folder (directory) in DOS, you have to type `mkdir C:\folder`. Then, to copy files to the new directory, you need to type `copy c:\file.dat c:\folder\file.dat` for each file. The Macintosh's interface is much easier to learn, remember, and use.

Why Use HyperCard as a Front End?

HyperCard is a perfect vehicle for front ends. The card and stack structure makes it simple to plan front ends. Each card represents an area that the system that is being front-ended provides. For example, two distinct types of areas are within CompuServe: services and forums. Services include electronic mail, known as EasyPlex,

and electronic shopping. Forums are an area where people with similar interests "meet" and exchange files and messages. For example, Apphyper is the forum for HyperCard enthusiasts. In a HyperCard front end, the stack contains separate cards for each service. Each forum shares a common background because they share functionality, but each forum retains its own card for ease of structuring (Figure 15-1).

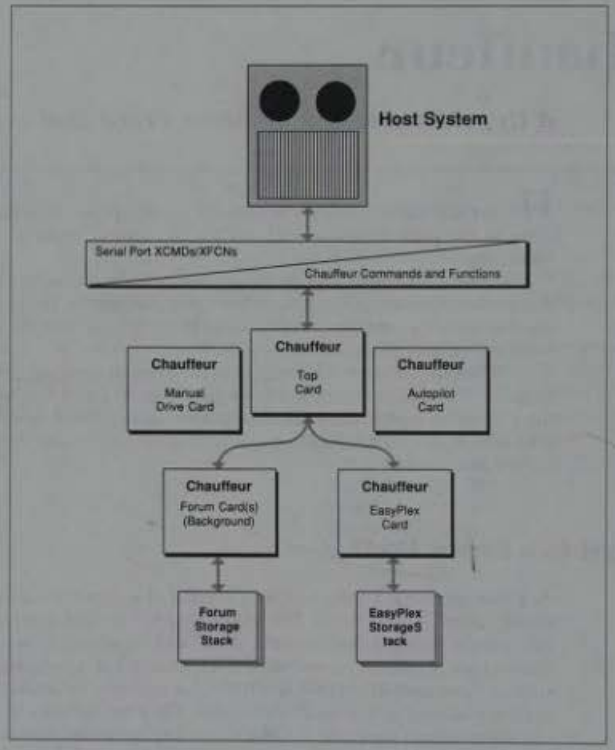


Figure 15-1. Cards in Chauffeur's stacks represent forums in CompuServe.

Extra cards and backgrounds can be created with a minimum of fuss in HyperCard. New services in CompuServe can be accommodated easily in Chauffeur using existing commands and functions. Because HyperCard uses an interpretive language, HyperTalk, Chauffeur can be modified effortlessly at a later stage. This is important when the front end is used in volatile environments where users can request modifications daily.

Graphics and sound form a major part of the HyperCard environment. Graphics can enhance the front-end interface and provide aids to complex tasks. Although sampled sounds require a lot of storage space, they can supply another way to prompt and inform users.

For operations that have to be fast, the advanced author can create XCMDs in Pascal or C. The commands and functions that Chauffeur uses are external, because HyperCard comes with no built-in commands or functions for communication through the serial port. Harry Chesley, Apple employee and author of The Waite Group's *Supercharging C* (Addison-Wesley, 1987) has written a set of XCMDS and XFCNS for serial port communication.

Because HyperTalk can be sluggish in handling text, you must be careful when you design the text aspects of front ending. Many algorithms and scripts that Chauffeur uses were refined several times. Front ends for communications systems should be fast because the user is paying for connect time.

General Front End Design

Chauffeur is a CompuServe-specific front end. By using the basic techniques described in this chapter, you can simplify the process of developing communications system front ends. Before you begin, however, there are a few important considerations.

First, the users of the system are the center of the developmental process. What they request is what you will try to deliver. Talk to the users and make a list of their requests. Because HyperCard is designed for instant, "on-the-fly" modification, you can make changes as you develop your stack.

Almost as important as what the user requests is the hallowed user interface. If users are intimidated by what is happening when they try to use your stack, your front end hasn't improved the situation. Sit several end users down with your stack and let them play. Watch their reactions to alerts; take note of their criticisms. Be receptive to requests for changes in design. Chauffeur passed through five iterations before reaching its current state, and I don't consider it finished yet.

You also must become familiar with the target or host system in its native environment. The target system for Chauffeur is CompuServe. You can get to know most communications systems by using standard communications packages such as Red Ryder or Microphone. It is essential to know the structure of the host system's menus and prompts and the different responses that may be returned. In becoming familiar with the host system, you may discover options, such as stringing commands together, that will enhance the speed of your front end.

While you are becoming familiar with the system, capture a transcript of the communications sessions. These transcripts are invaluable in the testing stages of your front end: the transcript can be used to emulate the host system without running up a hefty bill. Also, you can use a second Macintosh connected with a simple printer cable and communications package to the Macintosh that is running your stack to simulate the process and debug your stack without running up expensive bills.

What Is Chauffeur?

Chauffeur, written in HyperCard, is an interactive and self-running front end to CompuServe. Chauffeur derives its name from the idea that to get anywhere with a luxury vehicle, a chauffeur will add an atmosphere of opulence.

Figure 15-2 illustrates CompuServe running under Chauffeur. If you are familiar with CompuServe's interface (and we assume that you are), you see what an improvement this is!

The Chauffeur system consists of multiple stacks. The main controlling stack, called Chauffeur, holds all the scripts, cards, and buttons that facilitate the communications session. Chauffeur requires Hypercard version 1.2.1 or later.

There are also associated storage stacks. Various stacks contain all the accumulated messages to be sent and received. Each forum and the EasyPlex electronic mail system "owns" its own storage stack. Because it is difficult to remember the numbers assigned to each CompuServe user, Chauffeur also has a special stack where the user can store commonly used CompuServe addresses.

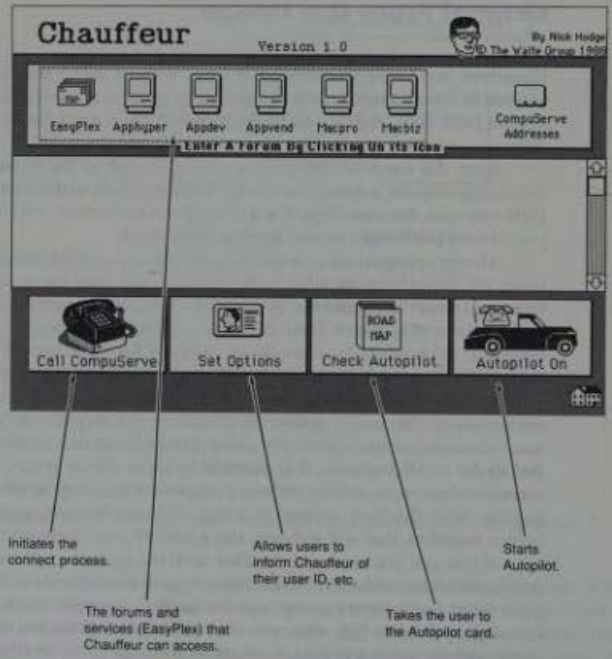


Figure 15-2. Running under Chauffeur, the CompuServe interface becomes easy to use.

The Chauffeur Help stack contains an easy-to-use help system that ties in with the main controlling stack. By holding down the Option key while clicking on an item in the Chauffeur stack, you are presented with a card displaying help information on that item.

The first card presented is the Top card (Figure 15-2), the home card of the Chauffeur system. From this card, the user can access each forum and service that Chauffeur enters.

Setting Options

Before you log on to CompuServe, you set options that you want to use. By clicking on the Set Options button, Chauffeur presents you with a set of questions: user name, user ID, phone number, and so forth.

After the options are set, you can log on to CompuServe. During this process, Chauffeur supplies CompuServe with your name and password and enters the Top menu. While you are online, Chauffeur changes the gray background with a more noticeable white background. This background, accompanied with a button flashing *online*, reminds you that you are spending money!

Using Chauffeur as a normal HyperCard stack is called *manual drive*. Users are in total control of where they are heading. They can respond interactively with CompuServe at any stage. On Autopilot, Chauffeur takes control of the communications session and follows a script taught by the user.

With Chauffeur's Autopilot option, you can program an Autopilot script for Chauffeur to follow. From the Top card, just put the Autopilot into operation and go and have coffee. Chauffeur handles all the work of logging on, entering EasyPlex, and reading and sending messages (Figure 15-3).

Another button that appears on all cards while you are online is the Hangup button. This button permits you to log off CompuServe at any time.

CompuServe Services

When you click on an icon, Chauffeur advances to that card. For example, if you click on the EasyPlex mail system icon, Chauffeur takes you to EasyPlex, both on-screen and in the transmission that is going on in the background.

At the EasyPlex card, you can send or read mail (Figure 15-4). If there are pending messages for you in EasyPlex, Chauffeur displays the number of unread messages underneath the read messages button. You can return to the Top card by clicking on the Top Menu button.

Forums

At the Top card, you can click on one of the forum icons and be presented with a forum (Figure 15-5). If there are any messages for you in that forum, Chauffeur places a message under the read messages button.

When entering a forum, Chauffeur first checks if you have joined the forum

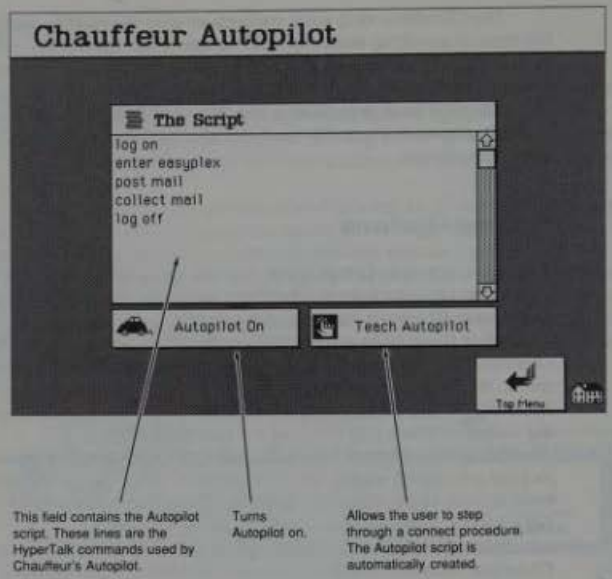

Chauffeur Autopilot
The Script
log on
enter easyplex
post mail
collect mail
log off
Autopilot On
Teach Autopilot
Top Menu
This field contains the Autopilot script. These lines are the HyperTalk commands used by Chauffeur's Autopilot.
Turns Autopilot on.
Allows the user to step through a connect procedure. The Autopilot script is automatically created.

Figure 15-3. Autopilot follows the scripts you write.

previously. If so, it continues; otherwise, Chauffeur gives you the option of joining the forum as a full member.

Next, Chauffeur checks if there are any members in conference. Again, Chauffeur supplies all of the responses to the menus that CompuServe presents.

There are more things to do than read messages sent to you and send messages to others. You also can read messages left in the forum for other people. You can get a listing of all messages in that forum and choose which ones you want to read.

Within a forum are different subtopics. From a pop-up field, you can set the subtopics that interest you. Only message headings from those subtopics are returned. The scanned listing is contained in the forum's storage stack and can be used to select messages to be read the next time you are online. You can read either an individual message or a thread from a message. (A thread contains the message itself plus any replies left in that forum.)

Modes

The transparency of the user interface is a major feature of Chauffeur. When Chauffeur is offline, all buttons that you use when online are still functional, except they show the stored data or allow you to compose messages to send. By using the buttons offline, users can learn the system's basics at their leisure--and much more cheaply than experimenting online.

Chauffeur's Internals

Chauffeur has been written to be open ended, because CompuServe is a varying system that is always being updated. Many functions can be changed with the end user noticing little.

But HyperTalk Has No Serial Port XCMDs

In HyperCard 1.2.1, there are no commands or functions that allow serial port communication to the outside world. These serial ports, called the printer and modem

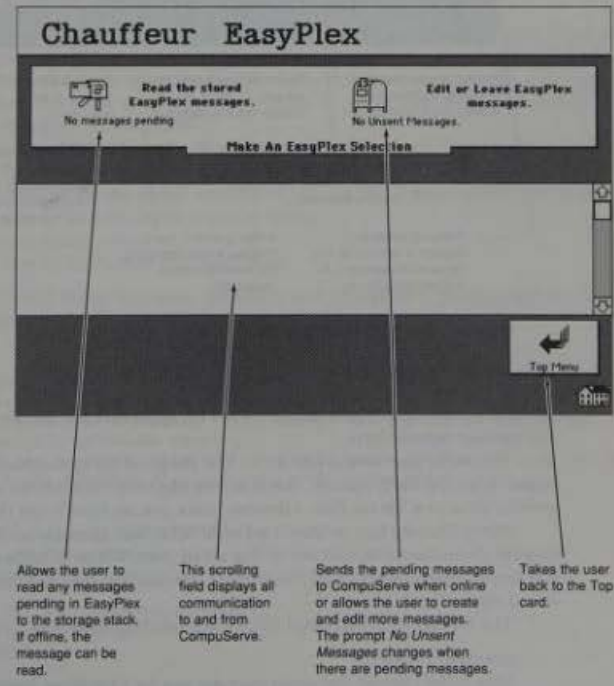

Chauffeur EasyPlex
Read the stored EasyPlex messages.
No messages pending
Edit or Leave EasyPlex messages.
No Unsent Messages
Make An EasyPlex Selection
Top Menu
Allows the user to read any messages pending in EasyPlex to the storage stack. If offline, the message can be read.
This scrolling field displays all communication to and from CompuServe.
Sends the pending messages to CompuServe when online or allows the user to create and edit more messages. The prompt *No Unsent Messages* changes when there are pending messages.
Takes the user back to the Top card.

Figure 15-4. With Chauffeur's EasyPlex card, you can send and read electronic mail.

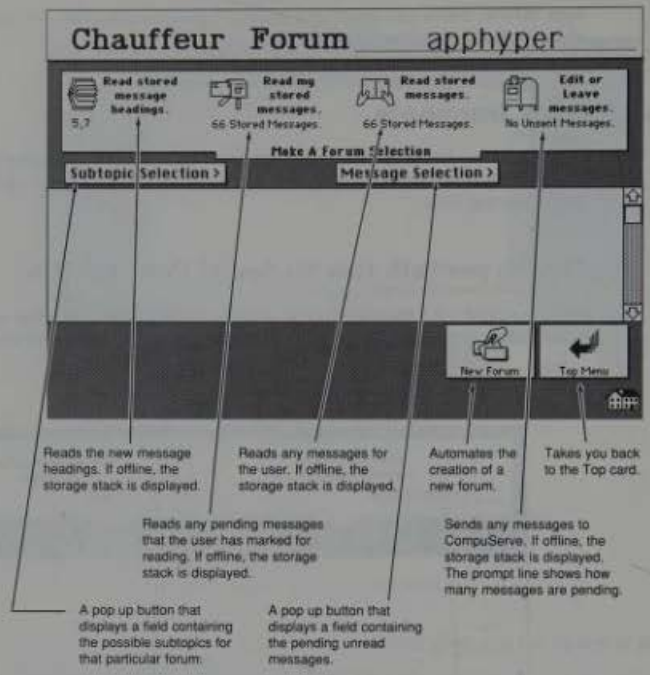

Chauffeur Forum apphyper
Read stored message headings.
5,7
Read my stored messages.
66 Stored Messages.
Read stored messages.
66 Stored Messages.
Edit or Leave messages.
No Unsent Messages.
Make A Forum Selection
Subtopic Selection >
Message Selection >
New Forum
Top Menu
Reads the new message headings. If offline, the storage stack is displayed.
Reads any messages for the user. If offline, the storage stack is displayed.
Automates the creation of a new forum.
Takes you back to the Top card.
Reads any pending messages that the user has marked for reading. If offline, the storage stack is displayed.
Sends any messages to CompuServe. If offline, the storage stack is displayed. The prompt line shows how many messages are pending.
A pop up button that displays a field containing the possible subtopics for that particular forum.
A pop up button that displays a field containing the pending unread messages.

Figure 15-5. The background card for the Apphyper forum.

ports, are used by modems and are used by AppleTalk for network communication and printer connections.

The serial port works like a file. The file must be opened before any work can begin. After the file is opened, the program can send data out the serial port just like writing data to a file on disk. Likewise, data can be read from the serial port.

Harry Chesley has written a set of XCMDs that allow HyperTalk programmers to send characters into and out of the serial port. These XCMDs are available from the Apple Programmers and Developers Association and are contained in a stack called HyperTerm.

The following XCMDs and XFCNs are used in Chauffeur:

`openSPort <port>`
This XCMD opens the serial port for use by Chauffeur before communication can start. The *port* variable is the serial port to use. It should be 1 for the modem port or 2 for the printer port.

`resetSPort <port,configuration>`
This XCMD resets the serial port *port* with the settings as defined in the *configuration* variable. Refer to *Inside Macintosh*, volume II, pages 250-51 for a full description of the settings for *configuration*.

`sendSPort <port,addLineFeeds,string>`
An XCMD that sends a string of characters held by *string* out of serial port *port*, adding line feeds after carriage returns if *addLineFeeds* is true. This command is used to take a command or message from HyperCard and send it out through the serial port.

`SPortHasChar <port>`
This XFCN returns true if serial port *port* has an unread character waiting. This is used as a check to see if there are pending characters before interrupting the flow.

`recvSPort <port>`
This XCFN returns the next character from serial port *port* if available or waits for the next available character.

`recvUpTo <port,termCharacter,timeOut,echo,edit>`
An XCFN that returns characters available up to *termCharacter* from serial port *port* or waits for *timeOut* 60ths of a second and returns those characters. If *echo* is true, *recvUpTo* echoes the characters out the serial port, adding line feeds after carriage returns. If *edit* is true, *recvUpTo* deletes the last character in the buffer.

`bufferSPort <port,oldBuffer,newBufferSize>`
This XCFN is used to change the size of the serial port *port* buffer. The *oldBuffer* is the global variable of the old serial port buffer. The *newBufferSize* is the new buffer's size in bytes.

`closeSPort <port>`
This XCMD closes the serial port.

Two of these externals, `openSPort` and `closeSPort`, were written by the author in LightSpeed C.

```
/*
openSPort XCMD
Written by Nick Hodge for Chauffeur.
Translated APDA MPW Pascal XCMD, except for the use of RAM serial
drivers and handshake setting.
Takes one parameter: the serial port to open. It then sets up the serial
port to do XON/XOFF flow control, both in and out.
*/

#include <SerialDvr.h>
#include <HyperXCMD.h>
#include <OSUtil.h>

void Failed (paramPtr,theError)
XCmdBlockPtr paramPtr;
int theError;
{
  Str255 tmpStr;
```

continued

```
  SysBeep (5);
  NumToString(theError,tmpStr);
  paramPtr->returnValue =
PasToZero(paramPtr,(StringPtr)tmpStr);
  /* mayday code! */
} /* Failed */

pascal void
main(paramPtr)
XCmdBlockPtr  paramPtr;
{
  short serialPort;
  int sPortInDvr,sPortOutDvr,OSError;
  Str31 str;
  SerShk *handShake;
  Handle firstPart;

  firstPart = paramPtr->params[0];   /* Grab first parameter */

  ZeroToPas(paramPtr,*(unsigned char **)firstPart,str);
    /* convert to Pascal string */
  serialPort = StrToNum(paramPtr,str);
    /* convert to number that is serial port # */

  if (serialPort == 2) {
    sPortInDvr = -8;
    sPortOutDvr = -9;
    OSError = RAMSDOpen(sPortB);   /* open the printer port */
  } else {
    sPortInDvr = -6;
    sPortOutDvr = -7;
    OSError = RAMSDOpen(sPortA);   /* open the modem port */
  } /* if */

  if (OSError != 0) {
    Failed (OSError);      /* whoops, something is wrong */
  } else {
    handShake->fXON = (char) true;
    handShake->fInX = (char) true;
    handShake->xOn = 0x11;    /* hex for XON character */
    handShake->xOff = 0x13;   /* hex for XOFF character */
    handShake->fCTS = (char) false;
    handShake->fDTR = (char) false;
    handShake->errs = 0;
    handShake->evts = 0;
    OSError = SerHShake (sPortInDvr,handShake);
   /* using the structure, set handshake */
    OSError = SerHShake (sPortOutDvr,handShake);
   /* for both incoming and outgoing comms.*/
    if (OSError != 0) {
   Failed (OSError);      /* whoops, something is wrong */
   } /* if */
   } /* if */
}  /* main */
```

Next, the CloseSPort XCMD:

continued


```
/*
closeSPort XCMD
Written by Nick Hodge for Chauffeur.
Translated APDA MPW Pascal XCMD, except for the use of the RAM serial
drivers.
Takes one parameter: the serial port to open. It then simply closes it.
*/

#include <SerialDvr.h>
#include <HyperXCMD.h>
#include <OSUtil.h>

#define nil 0

void Failed (paramPtr,theError)
XCmdBlockPtr paramPtr;
int theError;
{
  Str255 tmpStr;
  SysBeep (5);
  NumToString(theError,tmpStr);
  paramPtr->returnValue = PasToZero(paramPtr,(StringPtr)tmpStr);
    /* mayday code! */
} /* Failed */

pascal void
main(paramPtr)
XCmdBlockPtr  paramPtr;
{
  short serialPort;
  int OSError;
  Str31 str;
  Handle firstPart;

  OSError = 0;
  firstPart = paramPtr->params[0]; /* Grab the first parameter */
  ZeroToPas(paramPtr,*(unsigned char **)firstPart,str);
    /* convert to Pascal string */
  serialPort = StrToNum(paramPtr,str);
    /* convert to number that is serial port # */

  if (serialPort == 2) {
    RAMSDClose(sPortB);   /* close the printer port */
  } else {
    RAMSDClose(sPortA);   /* close the printer port */
  } /* if */

  if (OSError != 0) {
    Failed(paramPtr,OSError);  /* whoops, something is wrong */
  }  /* if */
} /* main */
```

Why rewrite these two externals, you may ask? During the testing phase of Chauffeur, there were problems with losing characters. This was blamed on the fact that the serial port was not handshaking—a technique used in communications to ensure that data is being received correctly. If the guest is running behind in collecting incoming characters, it tells the host to hold on for a minute while it catches up.

An XOFF character (Control-S) is sent to the host to tell it to halt for a while. After the guest has caught up, an XON character (Control-Q) is sent to reinitiate the communications session.

These externals open the serial port for use and set up the port to handshake. The `closeSPort` external closes the opened serial port driver. For further explanation, refer to *Inside Macintosh*, volume II, pages 250 and 251.

Using XCMDs and XFCNs as foundations, the structure of the front end can be built. These extra commands and functions can be written in HyperTalk or in a lower-level language such as Pascal or C. To facilitate easier understanding, HyperTalk is the preferred method. Although this results in slower execution, the script and structure are easier to decipher. Chauffeur uses many multifunctional scripts, all of which handle the communication that occurs between the front end and the host system.

The Three Types of Data in Communications

Essentially, there are three types of incoming data: prompts, menus, and messages. *Prompts* are lines of text sent out by the host that require the guest to read and type some sort of command. A *menu* contains lines of text that inform the user of the possible commands that can be typed at a prompt line. Sometimes a menu contains data that the front end must extract and use on the currently visible card. A *message* is any data that the front end must collect for the user to read at some later time.

Let's look at the case in which the front end is waiting for a prompt line before continuing. Chauffeur must read each line of data coming in and examine its contents against a set of expected values. Incoming data up to that line is discarded. A function handler called WaitFor waits a certain amount of seconds for possible prompts. The WaitFor handler can compare multiple prompts to the incoming data.

WaitFor is based on another function called IncomingChars. This handler is written in HyperTalk, but in the future it may be rewritten in C to increase the execution rate. Because this handler is a time-critical part of Chauffeur, it is optimized for maximum throughput. (Numbers in the listing relate to the author's comments in the text.)

```
function IncomingChars timeOut
  -- This procedure discards incoming text until it gets a match to
  -- "string" and returns the captured text and returns from
  -- back...
  global connected,searchString,returnString
  put 0 into whichMatch -- no match yet
  put false into ending -- no, we are not ending yet
  put 0 into ticksTaken -- and we haven't started yet
  multiply timeOut by 60 -- change to ticks, not seconds
  put empty into returnString
  put space into lastChars
  repeat with i = 1 to the number of items in searchString
    put last char of item i of searchString after lastChars
  end repeat
  repeat while not ending
```

continued


```
1   put recvUpTo(1,return,60,false,true) into incoming
    BeachBall
    if incoming is not empty then
2     put incoming after returnString -- and store for the return
      if lastChars contains last char of last line of ¬
      returnString then
        repeat with i = 1 to the number of items in searchString
          -- look for the matches
          if last line of returnString contains item i of ¬
          searchString then
3           put true into ending -- found one!
            put i into whichMatch -- we found this one (i)
            exit repeat
          end if
        end repeat
      end if
4   else
5     if not SPortHasChar (1) then -- no, just a slow comms speed
      add 60 to ticksTaken -- not found, so increment timeOut
        if ticksTaken > timeOut then
          put true into ending -- waiting too long
          -- now you do advanced checking, just in case
          get the number of lines in returnString
          get line it - 1 to it of returnString
          -- if no more chars, must be prompt
6         if lastChars contains last char of it then
            -- you have a prompt character, must be garbled prompt
            put 0 into whichMatch -- doesn't recognize the prompt
            CommProblems searchString -- tell user what is happening
          end if
7         if it contains "NO CARRIER" then -- CompuServe dumped us
            Hangup
            answer "The next message will cancel all scripts." with "OK"
            do "get the hilite of card field comms"
            -- force error to exit all
          end if
        end if
      end if
    end if
  end repeat
8 Display returnString  -- display incoming text
  if whichMatch is not 0 then delete last line of returnString
  return whichMatch  -- return what is found
end IncomingChars
```

IncomingChars reads the serial port and captures any incoming data up until a return (1) or for one second, whichever comes first. See Figure 15-6. If the incoming data returned by the `recvUpTo` XFCN is not empty, the script starts to check the lines of text it just received (2). If there is a match (3) with the desired list of responses, IncomingChars returns back to the calling handler. All data received in this process is stored into a global variable that any level handler can examine.

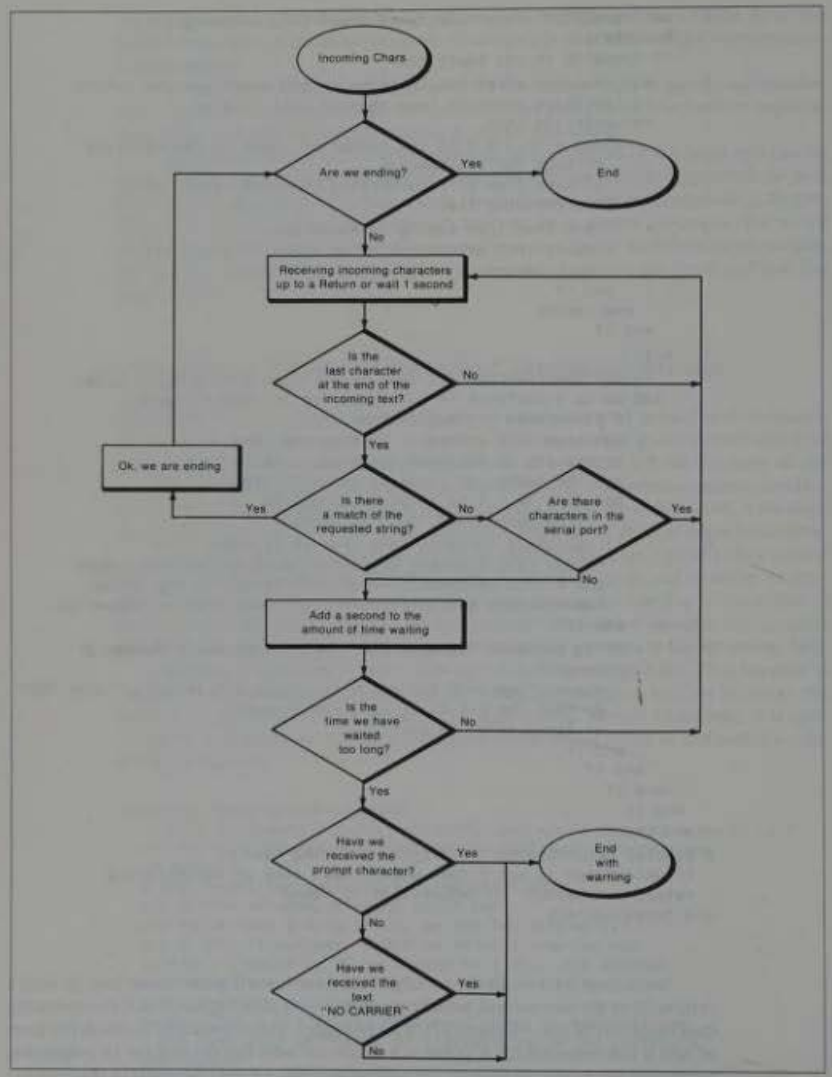


Figure 15-6. This flowchart details IncomingChar's functions.

Is Chauffeur Still Online?

In addition to looking for a match in the incoming data, the IncomingChars handler must make sure that the host is still talking to Chauffeur. The `ticksTaken` variable is incremented each time the `recvUpTo` function waits without any captured data (4).

Chauffeur first checks to see if the serial port was too slow in receiving some characters (5). It also checks the last three lines of captured data and looks for the prompt character (6), which is the last character in the line that is being searched for. In CompuServe, the prompt character is usually an exclamation mark or a colon. Chauffeur then uses a handler called CommProblems to inform the user that there may be a problem in the communications session. There is the possibility that CompuServe "dumped" or hung up. This is checked at 7. Finally, at 8, the incoming data is displayed in a scrolling field.

Receiving Data

Now let's look at receiving messages and real data. In CompuServe, data can be sent in two ways: paged and unpaged. Paged data is usually divided by prompt lines such as *Press Enter to continue!* Chauffeur automatically sends out a Return character when it detects this occurrence.

For paged data, a function handler called StoreUntilPaged is used. It takes three parameters: `continueString` is the prompt line that divides the message, and `timeOut` and `endString` both function in the same fashion as the WaitFor handler. StoreUntilPaged uses IncomingChars to save time at some later date.

Unpaged data is collected through the StoreUntil handler. Instead of using a `continueString` to indicate when to press Return, StoreUntil takes another parameter: `cancelString`. This parameter is a line that Chauffeur checks for when testing the incoming data to see if the host has finished sending messages. For example, when you are reading a threaded message in the forums, if there are no more messages in the thread, CompuServe returns to the *Function:* prompt. In the scripts for reading messages in forums, the line reads

```
put StoreUntil ("Function:",20,"Read action:") into forumMessages
```

The method that CompuServe uses to send out the messages follows. (What the CompuServe user types is underlined.)

```
Function: rt onl 12345
-- Message number 12345
Read Action: n
-- Message number 12347
Read Action: n
Function:
```

Upon receiving the word *Function:*, Chauffeur issues the CompuServe command `rt onl 12345`, which means read only the thread of message 12345. Following this, StoreUntil is called and returns the contents of message 12345 back to the

calling script, which issues the `n` response (read next message in thread). StoreUntil is called again. It returns message 12347 to the calling script, and again the calling script issues the `n` response. This time, when StoreUntil is called, it returns the word `canceled`, signaling that there are no more messages.

Sending Data

Higher-level handlers are used to control incoming data, but manipulating outgoing data is easier. From a front end, two types of data go out: commands and messages. To send commands in Chauffeur, a command handler called SendLine sends a line of text out through the serial port to CompuServe.

The SendMessage handler sends whole messages out to CompuServe. These messages are either EasyPlex mail being sent to other users or messages in the forums. Luckily, CompuServe has a standard editor that is similar in both EasyPlex and the forums. This editor takes a line of text, or 80 characters, at a time. Inside the editor, a prompt stating the current line number of the message is always displayed with a colon prompt. Part of the SendMessage handler algorithm divides the outgoing data into 80-character pieces. These 80 characters are only sent after receiving the colon prompt.

Using these six handlers, the process of writing the communications sections of the script is greatly simplified. All that is needed are the prompt lines and responses that the host system will return.

How Does Chauffeur Log On?

Before any of your scripts can work, Chauffeur must log on to the CompuServe network. This process requires some knowledge of modems and how they work.

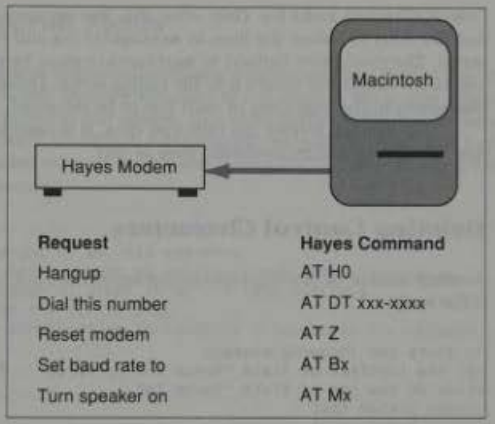
Modems have a "language," and one of the most popular languages is the Hayes command set. This command set has been adopted as an industry standard.

From the computer connected to the modem, the communications software sends a command prefixed with the two characters *AT*, which stand for *attention* (get ready for a command). The body of the command is followed by a Return character. After the connection is made, the modem sends a response—usually, *OK*—back to the connected computer.

The Hayes commands that Chauffeur uses are simple. Depending on what options you set up, Chauffeur sends a string of AT commands to the modem (Figure 15-7).

The autologon handler in the script of the Top card uses the set options to set the modem with these different Hayes commands. After a connection is established, Chauffeur gets the attention of the computer host at the other end and checks for the response. If the text line received by the `WaitFor` command is not a standard CompuServe *User ID:* prompt, it sends out the line *cis*, informing the gateway to "log me into CompuServe, please." (Gateways, such as TymeNet and satellites, usually redirect any communications to another system.)

When Chauffeur detects the *User ID:* prompt, it gives your user identification



Request	Hayes Command
Hangup	AT H0
Dial this number	AT DT xxx-xxxx
Reset modem	AT Z
Set baud rate to	AT Bx
Turn speaker on	AT Mx

Figure 15-7. The Hayes command set has been adopted as an industry standard.

to CompuServe. This is retained on disk after setting your user options. Following this, it provides your password at the *Password:* prompt.

Chauffeur has now logged onto CompuServe. It passes through the normal introductory messages and enters the Top menu. If you have set an autologon procedure within CompuServe itself, Chauffeur traps this and manually reenters the Top menu. At the Top menu, Chauffeur sets the CompuServe options to *set terminal other*, which makes sure that only the text is sent. Other terminal command set control characters are not sent; these control characters are used by special terminals such as Digital VT100s and other front ends for special formatting on the screen. After this, Chauffeur is online to CompuServe.

Removing Spurious Characters

In most communications hosts there are excess characters such as prompt lines, unwanted messages, headings, menu listings, and page numbers. These spurious chunks of text are detected by Chauffeur and removed by two special handlers.

The DeleteLFs handler takes a string of characters with linefeed characters and returns it without linefeed characters. Linefeed characters, sent by CompuServe at the beginning of each line, tell the guest communications program to advance by one line. For the technically minded, a linefeed character is a Control-J, ASCII 10. Almost every handler in Chauffeur dealing with raw incoming data calls DeleteLFs.

The other handler, DeleteUnWanted, has many purposes. It takes four parameters: `messageString`, `onlyAfter`, `mustContain`, and `addThis`. The `MessageString` container is the raw data. The `OnlyAfter` variable is a chunk of text that

DeleteUnWanted looks for. Only after this line appears will the DeleteUnwanted handler start to collate the lines in `messageString` and return them to the calling script. The characters defined in `mustContain` must be contained in a line before DeleteUnWanted will return it to the calling script. DeleteUnWanted adds `addThis` characters to the beginning of each line to be returned.

This handler refines the captured data. It is used when a captured message may be surrounded by irrelevant lines of text.

Deleting Control Characters

Another method of deleting control characters used in Chauffeur is a sly trick. Here is the script:

```
-- store the incoming message
set the lockText of field "forum In" to false
click at the loc of field "forum In"
choose button tool
choose browse tool
set the lockText of field "forum In" to true
-- and continue
```

This script utilizes a little-known anomaly in HyperCard: if you are editing a field and change from the browse tool to the button tool and back, all control characters in the active field are deleted. The script accomplishes the same under HyperTalk control.

Keeping the User Aware

Keeping the user informed is an important part of writing front ends. Experienced users want to know that the system is working. Likewise, inexperienced users require feedback to reflect what is going on. Chauffeur uses five methods to keep the user aware: sound, central scroll fields, alerts, a problems handler, and a background change indicator.

Sound

Sound is an interesting method Chauffeur uses to keep the user informed. Different functions have a set of two tones, indicating that a process has been completed. A double tone indicates reaching a certain section: a service or the Top menu. A rising set of tones indicates logging onto CompuServe, and several sets of falling tones signal a finished task. (These tones were chosen by a frustrated musician, not a human interface expert.) The scripts use the `SoundAlert` command to handle using the sounds for user feedback.

Central Scroll Fields

Second, Chauffeur displays all incoming data in a scrolling field in the center third of each card. You can examine this field at any stage of the online session to review what has happened in that session. To control the fields, the Display handler is in the script of the stack. Chauffeur automatically rescrolls the field to its lowest position. The script, called reScroll, is common among all Chauffeur scrolling fields that take incoming characters:

```
on reScroll
  put textHeight of me into textHite
  put round ((the height of me)/textHite) into visLines
  put the number of lines in me + 1 into numLines
  if numLines > visLines then
    set the scroll of me to textHite * (numLines - visLines)
  end if
end reScroll
```

Alerts

The third method of user feedback is a shadow field called Alert. This field is usually hidden, but it is displayed when required. A line or two of text is put into this field informing the user that the current process may take a while and is still working. This field is also used by the `SendMessage` command to show the percentage of the current message that has been sent.

Problems

The CommProblems handler provides necessary feedback apparatus in Chauffeur. This procedure is called if Chauffeur misses a prompt or times out. CommProblems is usually called by the IncomingChars handler when it has trouble with the communications session with CompuServe. The user can manually bail out of trouble, continue regardless, or hang up.

Card Background Change

Finally, Chauffeur changes the background color of each card in synchronization with the connect status. When Chauffeur is connected to CompuServe, the background is white. If Chauffeur is not connected, the background is standard gray. This trick is achieved by displaying a gray pattern on the background. When Chauffeur is online to CompuServe, the script through the `ShowConnectStatus` command shows an opaque white field called Dummy Background. This field is over the entire card, thereby changing gray to white. The HyperTalk `hide background` command could not be used because any graphics on the background would also be hidden.

Chauffeur's Overall Structure

When Chauffeur changes the current card, it sends commands to CompuServe to switch the primary section. The obvious place to contain this initial communication is in the openCard handler.

Each CompuServe function, such as sending mail and reading mail within a service, is associated with a button. In Chauffeur, scripts that automate CompuServe functions are in mouseUp handlers. This structure is used in all openCard handlers and mouseUp handlers on the Top, EasyPlex, and forum background cards. Each of the handlers that deal with CompuServe has a similar structure:

```
on handler
if the optionKey is down then go to help system
if learnMode is on then add me to Autopilot Script
if connected then communicate to CompuServe
  else just update the screen
```

The `learnMode` global variable is set true when users teach Chauffeur the path they want the Autopilot to follow when automatically logging on. Each handler adds a line of `Autopilot Script` to a field on the Autopilot card corresponding to its function. The `connected` global variable is an important variable that is set true when Chauffeur is communicating with CompuServe.

Structure of EasyPlex

EasyPlex is CompuServe's electronic mail system. Chauffeur has an entire card for EasyPlex functions: reading EasyPlex messages and sending EasyPlex messages.

The EasyPlex card has an openCard handler that is executed each time the card is opened. If connected, Chauffeur sends the CompuServe command: `go easyplex`. When you are connected to CompuServe, this command changes the current section to EasyPlex. Upon entering EasyPlex, Chauffeur checks to see if there are any pending messages for you and displays the number of messages in a field under the read messages icon.

The following script completes these tasks:

```
get WaitFor (20,"Enter Choice !","EasyPlex!","or HELp!")
  put DeleteLFs(returnString) into EasyPlexMenu
  if it is 1 then
     if EasyPlexMenu contains "READ mail" then
       repeat with i = 1 to the number of lines in EasyPlexMenu
         if line i of EasyPlexMenu contains "READ mail" then
           put offset ("READ mail",line i of EasyPlexMenu) + 10 ¬
           into starting
           put offset ("pending",line i of EasyPlexMenu) + 8 into ending
           put char starting to ending of line i of EasyPlexMenu ¬
           into card field "Read EasyPlex Count"
         end if
       end repeat
```

continued


```
     end if
  else
     repeat with i = 1 to the number of lines in EasyPlexMenu
       if line i of EasyPlexMenu contains "message" then
         put line i of EasyPlexMenu into card field ¬
         "Read EasyPlex Count"
         exit repeat
       end if
     end repeat
  end if
SendLine "set mode command" & return
get WaitFor (10,"EasyPlex!")
```

The script must accommodate three types of menus and prompts that may be set. This is accomplished by using multiple results to look for in the `WaitFor` line. Following this, the resulting value is tested to check which prompt was found: Enter Choice!, EasyPlex! or Help!

There are two styles for showing the number of unread messages. The resulting value is displayed in the Read EasyPlex Count card field. Finally, Chauffeur sends the command `set mode command` to CompuServe, which tells EasyPlex to display the one-line prompt. This shortened prompt line increases speed. The last `WaitFor` contains that one-word prompt, EasyPlex!.

The EasyPlex storage stack is examined by the openCard handler of the EasyPlex card to see if there are any unsent messages. The number is placed into the field below the Send EasyPlex icon.

At this point, Chauffeur is into the EasyPlex section of CompuServe. From here, you can read messages that are waiting for you or send messages out to any other user. It is important to note that Chauffeur creates messages only offline and allows you to read messages only offline.

When you click on the Read EasyPlex button, Chauffeur reads pending messages that you will respond to later when you are disconnected. The script sends out options to set page off—that is, to display the entire message and not break it up with Press Enter to Continue! prompt lines. Using the `read all` command, CompuServe displays each message, one after the other. The script works like this:

```
SendLine "set page off" & return
get WaitFor (30,"!")
SendLine "read all" & return -- read the message
repeat while not ending
  put StoreUntil("EasyPlex!",15,"Action!") into EasyPlexIn
  if EasyPlexIn is "canceled" then
    put true into ending
  else
    put DeleteUnWanted(EasyPlexIn,"Date:",empty) into EasyPlexIn
    -- and store them in the EasyPlex Storage stack
```

Using the `StoreUntil` command, Chauffeur can read all the messages in EasyPlex and store them in your EasyPlex storage stack. DeleteUnWanted takes away any spurious lines of text before the *Date:* line contained in all messages.

The Send EasyPlex button is on the EasyPlex card. It reads the pending mes-

sage from the storage stack and sends it to EasyPlex. Chauffeur sets the `set edit line` option to on to use the line-number-prompt method.

```
-- get the pending unsent message...
SendLine "set edit line" & return
get WaitFor (30,"EasyPlex!")
SendLine "compose" & return  -- tell CS you are composing
get WaitFor (15,"(/EXIT when done)")
put "From:" && userName & return before EasyPlexOut
put "To:" && whoto & return before EasyPlexOut
put "Subject:" && subject & return before EasyPlexOut
SendMessage EasyPlexOut,30,"card field alert"
-- okay, go for it again
```

Generally, after you create a message, CompuServe displays a range of prompts such as Send Message To:. Waiting for each of these prompts uses valuable time. To increase efficiency, CompuServe allows the user to add the subject matter, who the message is for, and who it is from at the beginning of each message to avoid these prompts. Chauffeur adds these lines to the head of each message sent:

```
To:  who to
From: who from
Subj: the subject
```

After sending and reading messages, you can log off using the Hangup button or reenter the Top menu.

Structure of Forums

The structure of forums is similar to the structure in the EasyPlex section of Chauffeur. Because functions in forums are shared across all CompuServe forums, Chauffeur has a single background for all forums.

All scripts for forums are also in the background. This includes the scripts for the four functions used in forums: reading scanned message headings, reading individual messages, reading your pending messages, and sending messages. This allows you to modify and improve the stack at different stages.

Again, the openCard handler sets up the initial communication to each forum. The forum that Chauffeur addresses depends on the name of the card. For example, the card that deals with the Apphyper forum is called Apphyper.

When entering the forum, the openCard handler checks to see if the forum is "unjoined" and lets you join it. Each forum in CompuServe gives you the option of joining the group after reading certain disclaimers and laws governing its use. The forum is unjoined and you are a visitor or guest until you supply the forum with your name. The process here is a simple matter of examining the returned text for the string of text VISITORS MENU. This text only appears if the forum is unjoined.

As you saw previously, Chauffeur also checks if there are messages left for you by other members in that forum. If so, the script places prompting text under the

appropriate icon on the screen. A similar check is applied to see if there is a conference in progress within that forum. If so, you are given the opportunity of joining the conference.

Finally, the pop-up field that contains a listing of the forum's subtopics is also checked. If this field is empty, Chauffeur gets the latest listing of subtopics by using the UpDateSubTopics handler, which executes the following script:

```
-- okay, let's read the listing
-- subtopics field is empty
SendLine "op" & return -- going into options
get WaitFor (20,"Enter choice !","Function:")
SendLine "su" & return -- subtopic selection
get WaitFor (12,"SUBTOPIC SELECTION")
put StoreUntilPaged ("<CR> for more !",30,¬
"Enter choice  ","Function:") into subTopics
SendLine "t" & return -- return to top menu
get WaitFor (20,"Function:","Enter choice !")
-- and return
```

As in in the EasyPlex section of CompuServe, Chauffeur sets the options upon entering the section. Chauffeur sends out the following:

```
-- this sets the options (only S - this session)
SendLine "op;sm n;re l;ed linedit;um off;s" & return
get WaitFor (12,"Function:")
-- and also sets the subtopics to view
```

The line can be interpreted as follows:

`sm n;re l`	stop at the end of message number `re 1`
`ed linedit`	set the editor to line-number-prompted editor for the SendMessage command
`um off`	set menus off; just use single-line prompts
`s`	for this session only

By setting options for the session only, the user is not inconvenienced at a later stage by not getting a familiar response from CompuServe. This is an important point to remember.

When you click on the read message headings button, a *qsn*, or quick scanned listing, is returned. This scanned listing is stored in the forum's storage stack for later perusal. The script for this is uncomplicated:

```
-- warn user
SendLine "qsn" & return -- do CompuServe qsn
put StoreUntil ("Press <CR>",60,"Function:") into ScanListing
put DeleteUnWanted (scanListing,empty,":",space) into scanListing
-- and store it
```

DeleteUnWanted cleans up the resultant text before storing it in the storage stack.

Reading selected messages is also a simple task, as is reading messages left for

you. After you have selected the messages you want to read by clicking on the read messages button, Chauffeur reads and stores them. There are two methods of reading messages: individual and threaded. Chauffeur uses the same script for both methods, but sends two different commands:

```
-- set options; warn user
if char 1 of it is "o" then
  put "ri" into readType
else
  put "rt onl" into readType
end if
-- make the screen look nice
SendLine readType && thisMessage & return
repeat while not ending
  put StoreUntil ("Function:",20,"Read action:") into forumMessages
  if forumMessages is not "canceled" then
    put DeleteUnWanted (forumMessages,"#:") into forumMessages
    -- and store the messages
```

By economizing on scripts, execution and maintenance are more efficient.

Sending Messages

Sending messages to the forum is a simple task. The only anomaly is that a message may be a reply to an existing message. It is important from a CompuServe point of view that a reply be sent to CompuServe as a reply, not as a separate message. This helps other users in the forum read a thread of a particular message trail. The script follows:

```
-- get the message first
if word 1 of whoto is "reply" then -- is it a reply?
  SendLine "re" && word 2 of whoto & return
  -- this is a reply
  get WaitFor (15,"to end message.")
  SendMessage forumOut,30,"card field alert"
  -- fix the other prompts that follow
else
  SendLine "l" & return
  -- this time you are just leaving a message
  -- fill in some prompts first here
  get WaitFor (15,"to end message.")
  SendMessage forumOut,30,"card field alert"
  -- it is all done
```

So, after the message is sent into the forum, the thread is still intact because Chauffeur has replied to a specific message in that forum. Like using the EasyPlex card, after sending and reading messages, you can log off using the Hangup button or reenter the Top menu by clicking on the appropriate buttons.

Autopilot: How Does It Work?

A major feature of Chauffeur is the Autopilot. This interesting set of scripts allows Chauffeur to run the front end virtually by itself. This process can be implemented easily into any front end.

The central part of the Autopilot is the idle loop in the script of the stack. This loop executes a line of the Autopilot script if the Autopilot is turned on. The status of the Autopilot is contained in the `selfRunning` global variable. Here is the idle handler:

```
on idle
  global connected,selfRunning,executeLine,learnMode
  -- some modal button stuff
  if selfRunning then
    get line executeLine of card field "Chauffeur script" of ¬
    card "Chauffeur"
    if it is not empty and (char 1 of it ≠ "-") and ¬
    (char 2 of it ≠ "-") then
      do it
      put 1 + executeLine into executeLine
    else
      if connected then
        log off
      end if
    end if
  end if
  -- some other script stuff here
  pass idle
end idle
```

This handler contains the line `do it`. The HyperTalk `do` command is a powerful command. In this case, it is used to execute a line of a field of text as if it were a HyperTalk script. A set of commands that describe a particular task can be created: log on, collect mail, post messages, and so on. Chauffeur has a collection of these commands:

```
log [on|off]
enter [forum name|service name]
collect [mail|headings|messages]
post [mail]
```

Using these new commands, you can build an Autopilot script that is easy to read and maintain.

The scripts that handle the four commands are as follows:

```
on log onOff
  global selfRunning,csPosition,connected
  if onOff is "Off" then
    send mouseUp to bkgnd button "Hangup"
    enter "Top"
    put false into selfRunning
```

continued

```
  else
    enter "Top"
    send mouseUp to card button "Connect"
  end if
end log

on enter serviceName
  global csPosition
  visual effect dissolve
  go to card serviceName
  put the short name of this card into item 1 of csPosition
  put the short name of this background into item 2 of csPosition
end enter

on collect typeMessage
  global selfRunning,csPosition
  if selfRunning then
    if item 1 of csPosition is "EasyPlex" then
      send mouseUp to card button "Read EasyPlex"
    else if item 2 of csPosition is "forums" then
      repeat with pCount = 1 to the paramCount
        get param (pCount)
        if it is "headings" then
          send mouseUp to card button "QSN"
        else if it is "mail" then
          send mouseUp to card button "Read Mine"
        else if it is "messages" then
          send mouseUp to card button "Read These"
        end if
      end repeat
    end if
  end if
end collect

on post typeMessage
  global selfRunning,csPosition
  if selfRunning then
    if item 1 of csPosition is "EasyPlex" then
      send mouseUp to card button "Send EasyPlex"
    else if item 2 of csPosition is "forums" then
      send mouseUp to card button "Send Pending"
    end if
  end if
end post
```

Using the HyperTalk `send` command, you can simulate the user clicking on buttons and entering different cards. The scripts that handle the communications side of Chauffeur just check the `selfRunning` global variable to verify the self-running status before putting prompts on the screen.

The `post` and `collect` commands are context sensitive—that is, in different sections of CompuServe, they are required to send disparate commands to Chauffeur. Therefore, another global variable, `csPosition`, has been defined to allow each handler to check the context.

User Selections in Scrolling Fields

Each forum first supplies a quick scan listing of the messages it holds. After the listing is collected, Chauffeur allows you to select a message to be read when you are next online. The listing is in a scrolling field in the forum storage stack. Following is the script behind the field that allows you to click on a particular message. (Numbers relate to the author's comments in the text.)

```
on mouseUp
  global forumName,storageName
  lock screen
  set lockmessages to true
  set the cursor to watch
1 get item 2 of the clickLoc
  subtract the top of me from it
  divide it by the textHeight of me
  add (the scroll of me/ the textHeight of me) + 1 to it
  put line trunc (it) of me into messageNumber
  put trunc(it) into lineNumber
  if messageNumber is not empty then
    get word 1 of messageNumber
    if first character of it is not space then
      delete first character of it
    end if
    if last character of it is ":" then
      delete last character of it
    end if
    push card
    go to first card in background "Messages In"
2   find it in field "Message Number"
    if the result is "not found" then
      pop card -- return to listing
      pop card -- return to controller stack
      push recent card -- push the listing card
      if char 1 of messageNumber is space then
        delete first character of messageNumber
3       answer "Read in which manner?" with "Threaded" ¬
        or "Only"
        if it is "Threaded" then
          put "t" before messageNumber
        else
          put "o" before messageNumber
        end if
4       put messageNumber && return after card field ¬
        "Read These" of card forumName
        pop card -- return to listing
        push recent card
        -- this is the controller stack return place
        do "put messageNumber into line lineNumber of" && the ¬
        target
      else
5       delete first character of messageNumber
        go to card forumName
        repeat with i = 1 to the number of lines in card field ¬
```

continued

```
        "Read These"
           if messageNumber is in line i of card field ¬
           "Read These" then
              delete line i of card field "Read These"
              exit repeat
           end if
        end repeat
        pop card -- return to listing
        push recent card
        -- this is the controller stack return place
        do "put space & messageNumber into line lineNumber of" ¬
        && the target
      end if
    end if
  end if
  set lockmessages to false
  unlock screen
end mouseUp
```

Chauffeur first calculates the line number of the mouseClick (1). This line is extracted from the field and used by Chauffeur to derive a message number. Subsequently, Chauffeur attempts to find that message in the existing database of messages (2). If the result is not found, Chauffeur realizes it must be a message to be read when next online or a message you want to delete from the read list when next online.

If the message is being added to the read list, an answer alert box (3) allows you to specify if you want to read the message threaded or individually. The character *o* indicates only, or individual, and the character *t* indicates threaded. This message number is added to the pop-up field that appears on the forum card in Chauffeur (4). Otherwise, the message is deleted from the read list in Chauffeur (5).

These messages can now be read by Chauffeur when next online to CompuServe, using the first character, *o* or *t*, to indicate only or threaded.

Exploring Your Messages

After data is accumulated in the storage stacks, what then? Data by itself is simply a collection of words: what we need is a system in which correlations are obvious to the user.

In the forum storage stacks, Chauffeur has an interesting "hypermedia"-like technique of reviewing the messages. The script of the forum field follows:

```
on mouseUp
  lock screen
  get the clickLoc
  set the lockText of me to false
  click at it
  click at it
  put the selectedText into findThis
  put the short id of this card into currentCard
  set the lockText of me to true
  if it is not empty then
```

continued


```
    repeat
      find whole findThis in me
      if the result is not empty then
        beep
        exit repeat
      end if
      if not(the short id of this card is currentCard) then
        exit repeat
      end if
    end repeat
  end if
end mouseUp
```

This script monitors mouseUp messages. After getting a mouse click, it uses a devious method to find the word that was clicked on. It first sets the `lockText` property to false, allowing the text within the field to be edited, then double-clicks at the mouse position to return a single word within the `selectedText` property. This word is searched for elsewhere in the storage stack.

The concepts and techniques in this handler have been in the HyperCard community for some time. If your front end deals with mountains of incoming messages, it may be advantageous to include similar scripts.

Replying to messages can also be greatly simplified. Why retype the subject and recipient when it is already in the original message? In the EasyPlex stack, the following script is found in the Reply to this Message button:

```
on mouseUp
  repeat with i = 1 to the number of lines in field "EasyPlex In"
    get offset ("From:",line i of field "EasyPlex In")
    if it > 0 then
      put char it + 7 to length(line i of field "EasyPlex In") ¬
      + 1 of line i of field "EasyPlex In" into whoto
      put i+1 into i
      put line i of field "EasyPlex In" into subject
      delete first word of subject
      exit repeat
    end if
  end repeat
  LeaveMessage whoto,subject
end mouseUp
```

This script searches the inbound message line by line for the characters *From:*. In the structure of the incoming message, this line is followed by the line containing the subject. These two containers are passed to the LeaveMessage handler, which uses these two values to create a new outbound message, automatically inserting the subject and receiver.

Hiding Password Input

Some interesting handlers in the script of the Top card set options before Chauffeur connects to CompuServe. The SetOptions handler asks a series of questions regard-

ing user name, user ID, and so forth, and stores the responses in an options file. This file is used thereafter by Chauffeur to connect to CompuServe.

The EnterPassword handler is a little different. In communications systems, it is common to hide the entry of passwords. In HyperCard, a password entry cannot be hidden unless `blindTyping` is implemented so that you can type into a hidden Message box. Chauffeur uses this technique for blind input of a password. The scripts follow:

```
on EnterPassword
  global returnCount,firstPass,oldBlindTyping
  show card field "alert"
  put return & "Please type your CompuServe password." into ¬
  card field "alert"
  put the blindTyping into oldBlindTyping
  set blindTyping to true
  hide msg
  put 0 into returnCount
  put empty into firstPass
end EnterPassword

on returnKey
  global returnCount,firstPass,oldBlindTyping
  if returnCount is empty then
    pass returnKey
    exit returnKey
  end if
  if returnCount is 0 then
    put msg into firstPass
    put empty into msg
    put 1 into returnCount
    put return & "Please type your password in again to check." ¬
    into card field "alert"
    hide msg
    exit returnKey
  end if
  if returnCount is 1 then
    if msg is firstPass then
      hide msg
      hide card field "alert"
      set blindTyping to oldBlindTyping
      put empty into msg
      WriteOption firstPass
      close file "Chauffeur Options"
      put empty into firstPass
      put empty into returnCount
      hide msg
    else
      answer "Sorry, the password did not check out." with "OK"
      EnterPassword
    end if
  end if
end returnKey
```

First, the EnterPassword handler stores the old `blindTyping` setting in a global variable. There is nothing worse for HyperCard users than to have something changed without their knowledge. Always reset values like this to their original setting. A prompt is placed into the alert field to ask the user to type the password. After the Return key is pressed, the returnKey handler is called into action. It checks the number of times the Return key is pressed. If it is the first time, Chauffeur prompts the user to type the password in again to double check the original entry. The second time the Return key is pressed, if all is well, the password is saved to the options file.

Help!

Implementing the help system for Chauffeur was a simple task. In Chauffeur, the user just holds down the Option key and clicks on an item to display a help card.

A handler in the script of the Chauffeur stack channels all help calls:

```
on ChauffeurHelp object
  global connected
  if not connected then
    lock screen
    go to stack "Chauffeur Help"
    find object in field "keywords"
    unlock screen with dissolve
  else
    ConnectedWarning
  end if
end ChauffeurHelp
```

This handler is called from almost every script of an object in Chauffeur. The command takes one parameter, `object`, and opens the Help stack to the card displaying help. Nothing could be simpler.

In the script of the Chauffeur stack, another handler traps a mouseUp call in case the user wants help on a card or background:

```
on mouseUp
  if the optionKey is down then
    get the short name of this bkgnd
    if it is not "forums" then
      get the short name of this card
    end if
    ChauffeurHelp it
  end if
end mouseUp
```

Implementing a help system is as easy as typing text on some cards in a separate stack. If you establish a help system, your users will feel more comfortable with your front end.

HyperCard to the Rescue

HyperCard is a perfect environment for creating front ends for communications systems. One major deficiency in many communications services is the lack of graphics, which results in a user interface that is inferior to the Macintosh user interface. It is like giving people the user interface of the Macintosh, yet forcing them to use MS-DOS style commands.

With a little work, you can overlay a set of complex commands with a familiar interface. With a HyperCard front end, the users of your stack will be more at ease, more efficient, and more productive with your communications system. By using HyperCard and the methods described in this chapter, you can give a familiar Macintosh interface to the coldest communications system.

HAYDEN Macintosh Library BOOKS

Available on Disk

The Waite Group's Tricks of the HyperTalk™ Masters

Take full control of HyperTalk with this collection of tips, tricks, tools, tutorials, and techniques. This intermediate-to-advanced level book is for programmers, students, business people, and professionals who already know the basics of scripting in HyperTalk and want to go further.

Covering the International Version 1.2.2 of HyperCard, this book helps you do the seemingly impossible: make synchronized scrolling fields so that you can mix fonts and styles and create your own custom drop-down, pop-up, relocatable, tear-off menus, and moveable menu bars. Because these scripts (and most others in the book) are pure HyperTalk, you can type them immediately without external commands or functions. You will also discover details on creating speedy "sprite" animations that use fonts and give 32,000 "virtual" graphics layers, techniques for animating with buttons, and extensive information on creating music videos that use "paged" sound for synchronization.

Tools provided in *The Waite Group's Tricks of the HyperTalk Masters* include scripts for making multisided buttons that can cover irregular regions, a master template to create custom self-administering quizzes, and a pop-up window maker. Over a dozen utilities will help you keep track of and improve your scripting. Chauffeur, a HyperCard to CompuServe interface, demonstrates the ease of writing a HyperCard front end.

Developers will find tips for working with large stacks, a definitive chapter of HyperTalk benchmarks, and a compendium of ways to use the best external commands and functions available today. A final information-packed reference chapter lists every HyperTalk command, function, keyword, operator, and reserved word with multiple examples — in one place, more details than in all the official Apple documentation.

About the Authors

The Waite Group is a developer of computer, science, and technology books. Acknowledged as a leader in its field, The Waite Group creates book ideas, finds authors, and provides development support throughout the book cycle, including editing, reviewing, testing, and production control for each title. The Waite Group has produced over 70 titles, including such best-sellers as *C Primer Plus, Microsoft C Bible, MS-DOS Bible, Tricks of the MS-DOS Masters,* and *UNIX System V Primer.* The Waite Group produces 15 to 20 new computer books each year and has yearly sales of a half-million books. Mitchell Waite, president of The Waite Group, wrote his first computer book in 1976. The Waite Group can be reached at 100 Shoreline Highway, Building A, Suite 285, Mill Valley, CA 94941, 415/331-0575.

HAYDEN BOOKS
A Division of Howard W. Sams & Company
4300 West 62nd Street
Indianapolis, Indiana 46268 USA

$24.95 US/48431

ISBN 0-672-48431-5

9 780672 484315 90000